E2K - Архитектура и особенности

HiLoad 2021

Что такое Эльбрус

- Основан на разработках СССР, не клон зарубежного процессора
- Полностью спроектирован в России, есть модель, которая производится в России, в основном производится на TSMC
- Умеет исполнять код x86 в режиме двоичной трансляции, в остальном не имеет с x86 ничего общего
- Довольно производителен: «Эльбрус-16С» — 16 ядер, 2 ГГц, 750 Гфлоп/с, 16 нм
- Серийно производится с 2014 года и практически применяется
- ОС - несколько дистрибутивов Linux (Эльбрус, МСВС, ALT Linux, Astra Linux), сообщается, что существует версия QNX под Э2К.

История

- 1987 год: Эльбрус-1 - 15 млн флопс, 64 M RAM - опирался на опыт создания БЭСМ-6
- 1980 год: Эльбрус-2 - 8 процессоров, 125 млн флопс, 144 M RAM
- Эльбрус-3 начат в 1985 году, но запустить в серию не удалось в связи с распадом СССР. Единственный работающий экземпляр - 1994 год. Создатели Эльбруса начинают работать над процессорами Интел.
- Только в 2014 году выходит первый процессор серии Э2K - Эльбрус 4С. Тактовая частота 800 МГц, 50 Гфлопс. Эстеты делают фи - гадкий, негодный процессор.

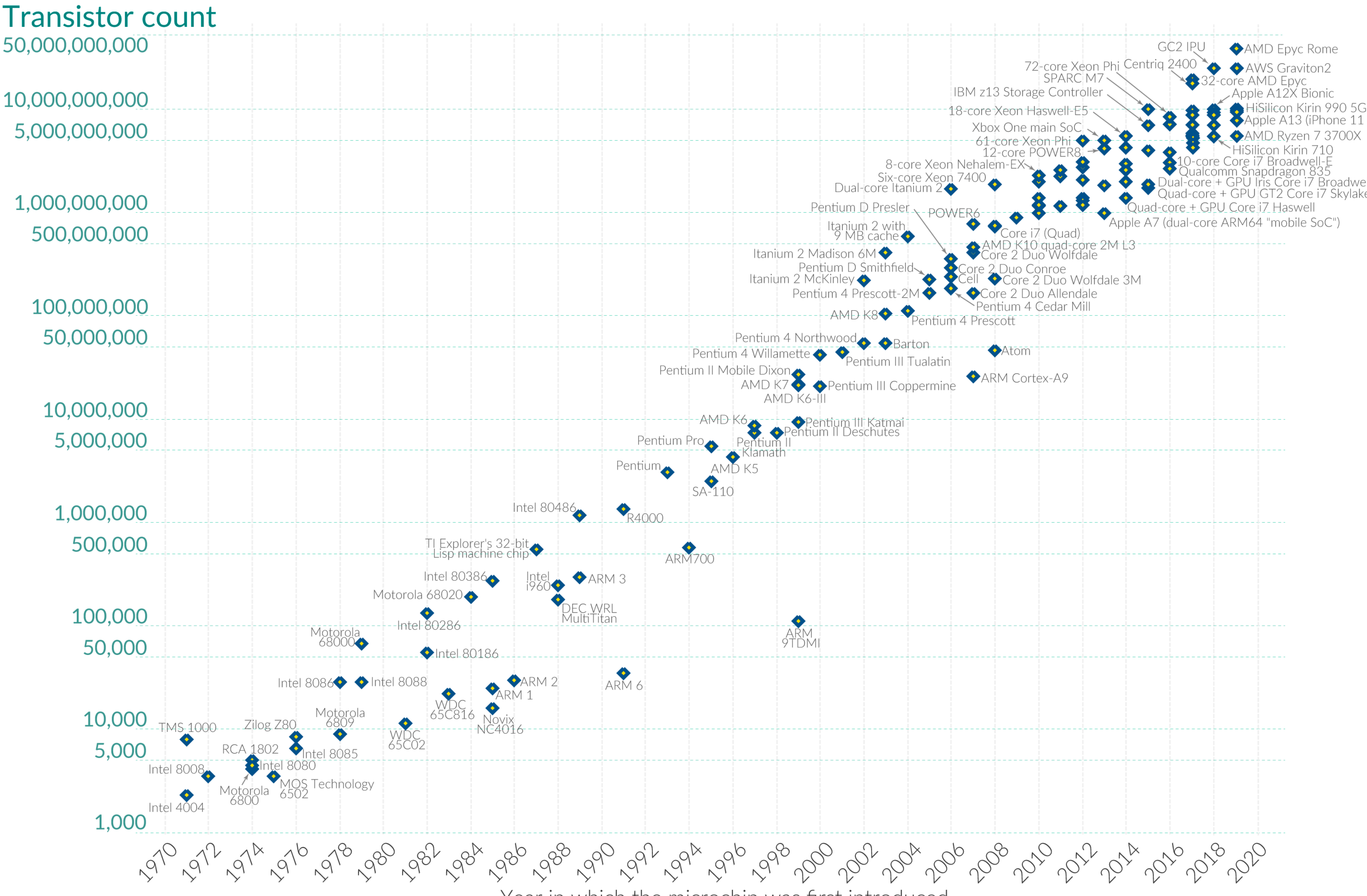
2021 год: Эльбрус 16С

- 16 ядер
- 2ГГц
- 1500 ГФлопс (32 бита)
- До 50 команд на такт
- до 64 ядер в SMP системе. Реально на Э-8С выпускались 4-процессорные модули, данных по Э16 пока нет, но 4 процессора = 64 ядра.
- Шина 200 Гбайт/с

Moore's Law: The number of transistors on microchips doubles every two years

Our World in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.



Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)

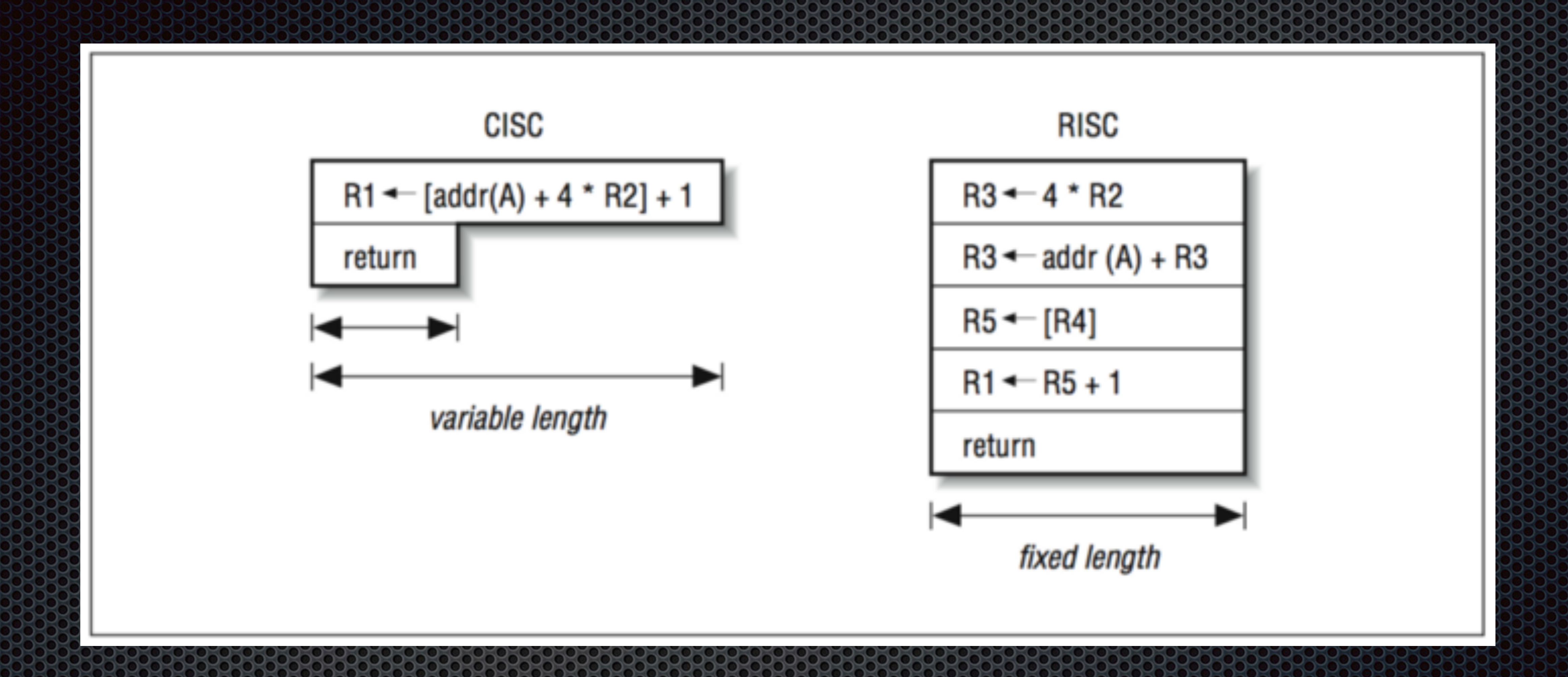
OurWorldinData.org – Research and data to make progress against the world's largest problems.

Закон мура

- Удвоение числа транзисторов на кристалле каждые 24 месяца.
- Закон работает до сих пор (см. график). Но!
- До 1995 года это удвоение давало прямой прирост производительности единственного ядра процессора, то есть ускоряло работу всех программ без исключения, и не требовало никаких усилий от программистов. Просто запусти программу на новом процессоре.
- Примерно с 95 года начались проблемы. Прирост производительности отдельного ядра начал буксовать. Процессоры пошли в сторону параллелизма.

Скрытый параллелизм

- Первые попытки были сделаны прозрачным для программиста образом - CISC процессоры архитектуры x86 переродились - теперь они CISC только снаружи, но RISC внутри.
- Крупные RISC-команды разбиваются на последовательности RISC-style внутренних инструкций процессора. Затем в потоке таких инструкций во время работы процессора выявляются независимые друг от друга участки, которые выполняются параллельно - внутри себя процессор фактически является многоядерным и способен выполнять несколько потоков инструкций.
- Такая модель позволяет (с известной вероятностью) ускорять программы не требуя никаких усилий от программиста и компилятора. За счёт кардинального усложнения процессора.


CISC
R1 ← [addr(A) + 4 * R2] + 1
return
variable length
RISC
R3 ← 4 * R2
R3 ← addr (A) + R3
R5 ← [R4]
R1 ← R5 + 1
return
fixed length

Явный параллелизм

- Фактически такое усложнение, конечно, привело и к усложнению компилятора - правильное расположение инструкций в коде упрощает работу процессору и повышает вероятность распараллеливания.
- В итоге это усложнение всё равно не решило всех проблем - современные процессоры явно многоядерные и требуют от программиста явного ручного распараллеливания алгоритмов. Хотя в пределах ядра описанная выше модель CISC/RISC преобразования на x86/x64 процессорах до сих пор работает.

VLIW: ручной параллелизм

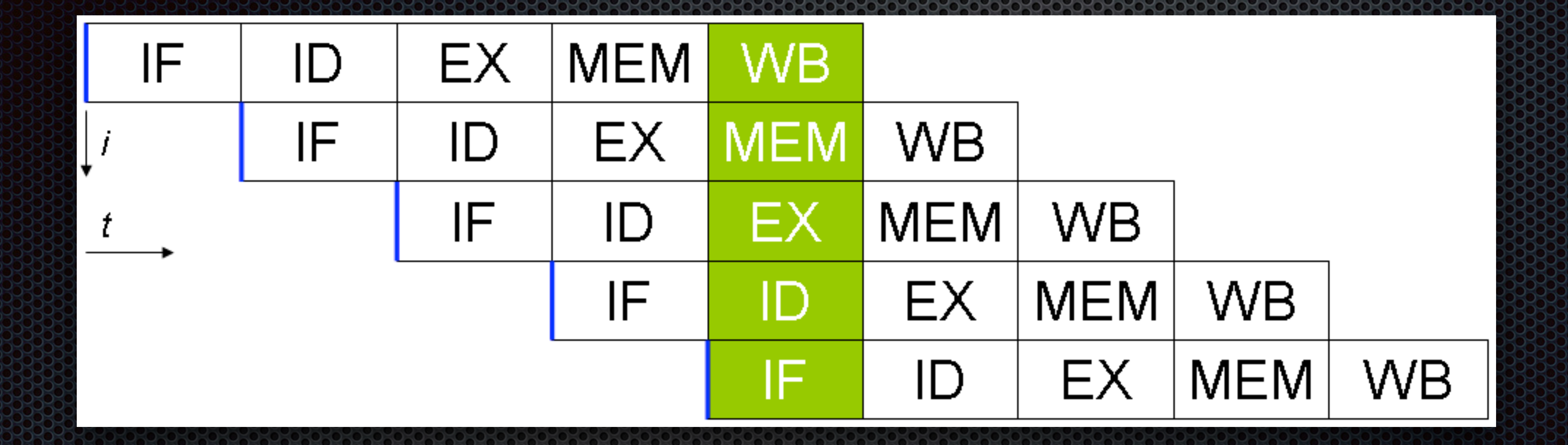
- При разработке Э2К был принят кардинально другой подход. Зачем усложнять кристалл, заставляя его распараллеливать изначально линейный код? Можно перенести эту работу полностью на компилятор. Он в любом случае знает о программе больше, чем процессор, может работать долго, анализировать крупные участки кода и принимать решения об оптимизации на более высоком уровне.
- Снижаем неопределённость - в обычных процессорах параллелизация - результат работы и компилятора, и процессора - и они должны понимать друг друга
- Инструкция (широкая команда) для Э2К содержит прямые и конкретные указания для всех вычислительных блоков процессора. Если что-то можно сделать параллельно - об этом процессору говорит компилятор.

На борту Эльбруса

- Каждое ядро процессора содержит шесть АЛУ, которые работают параллельно. Для каждого АЛУ в широкой команде есть свой фрагмент - слог.
- Кроме слогов для шести АЛУ широкая команда может содержать:
- Инструкцию для юнита передачи управления
- 3 вычисления на предикатах и 6 квалифицирующих предикатов
- 4 инструкции для асинхронного чтения данных в цикле
- 4 литерала в 32 бита

Предикаты

- Регистр предикатов: каждые 2 бита этого регистра рассматриваются как булево значение - предикат. Над битами в регистре можно выполнить логические операции, и затем - обусловить выполнение любого слога значением предиката. Второй бит позволяет предикату иметь “недопустимое” значение.
- Это позволяет упаковывать код типа “if(x < 0) x = 0; else x--;” в пару команд вообще без условных (и безусловных) переходов.



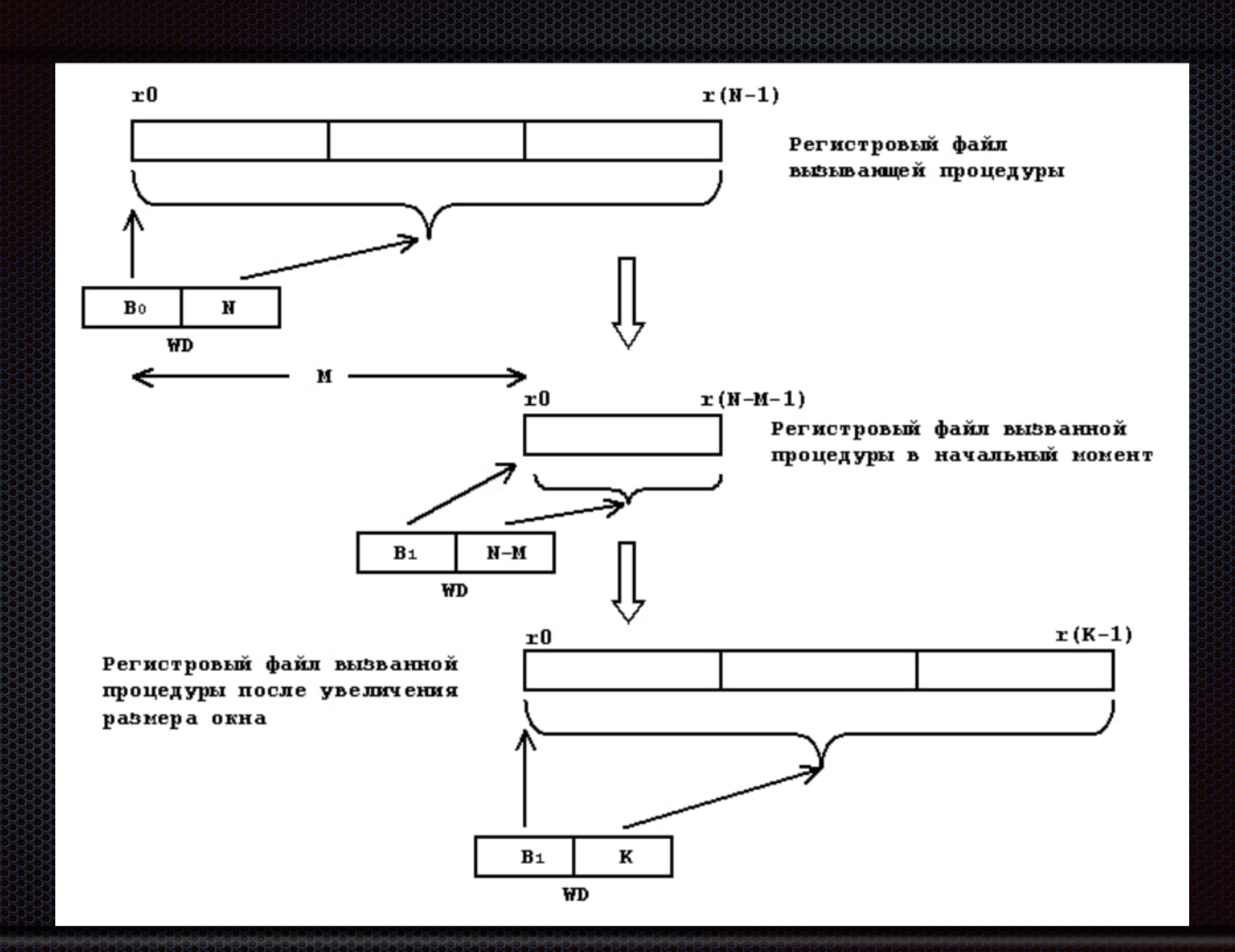
Команды перехода (jmp) - ад для любого процессора. Они сбивают работу конвейера и обходятся процессору как добрый десяток других команд.

Подготовка переходов

- В Э2К все переходы (включая вызовы и возвраты из подпрограмм) выполняются в два этапа. На первом этапе команда вычисляет адрес перехода. Эта команда может быть выполнена (запущена на исполнение) как угодно рано.
- На втором этапе переход выполняется. Если мы достаточно рано запустили команду вычисления адреса перехода, то команда собственно перехода будет выполнена абсолютно безболезненно, конвейер процессора не собъётся.
- Э2К позволяет подготавливать до трёх переходов одновременно (на случай, если мы пока не знаем, по какому пути пойдёт программа).

Регистровое окно

- Процессор содержит 256 84-разрядных регистров. При этом не все регистры могут быть видны программе - каждая функция “заказывает” себе у процессора нужное количество. Видимые функции регистры делятся на три последовательные группы.
- Регистры, в которых вызывающая функция передала нам параметры.
- Регистры, в которых мы просто работаем.
- Регистры, в которых мы будем передавать параметры вызываемой функции.
- В момент вызова функции регистровый файл виртуально “сдвигается” вниз - первая и вторая группы “уходят ниже ватерлинии” и не будут видны вызванной функции. Третья группа становится первой, а новые вторую и третью группы вызванная функция “заказывает” себе у процессора специальной командой.


r0
r(N-1)
Регистровый файл
вызывающей процедуры
B0
N
WD
M
r0
r(N-M-1)
Регистровый файл вызванной
процедуры в начальный момент
B1
N-M
WD
r0
r(K-1)
Регистровый файл вызванной
процедуры после увеличения
размера окна
B1
K
WD

Стек регистров

- Если в процессору хватило физических регистров для того, чтобы обеспечить аппетиты обоих (вызывающей и вызываемой) функций, то обращения к памяти вообще не происходит. Если функции жадные до регистров или глубина вложенности велика, и физические регистры кончились, процессор прозрачно для программы сохраняет “нижнюю” часть регистрового файла в стеке одной крупной операцией записи, что куда более эффективно, чем обслуживание мелких push/pop инструкций в традиционных архитектурах. Естественно, что по мере возвратов из функций процессор так же прозрачно для кода вернёт регистры из стека.
- Есть четвёртая группа регистров - глобальные. Они видны всем функциям, не сохраняются при вызове и, таким образом, могут быть использованы как временные внутри функций или как регистры доступа к глобальным данным.

Подкачка данных для цикла

- Процессор Э2К способен выполнять циклы с наложением итераций - одна или несколько итераций цикла могут начать исполнение до того, как предыдущая итерация завершилась.
- Для этого в регистровом (и предикатном) файле может быть организована последовательность (конвейер) регистров - несмотря на то, что код всех итераций будет обращаться к одному и тому же регистру, процессор будет выделять каждой новой запущеной итерации новый регистр из конвейера.
- Более того, отдельный параллельно работающий юнит загрузки данных для цикла будет подгружать данные из памяти по определённой программистом схеме. Этот юнит загружает данные в FIFO с опережением, а код цикла выбирает их из FIFO по мере выполнения итераций цикла. Это позволяет выполнять циклы не задерживая итерации на ожидание данных из памяти.

Несколько стеков

- Э2K поддерживает несколько параллельных стеков. В частности, сохранение адреса возврата и состояния процессора выполняется на своём стеке, а регистров пользователя - на своём. Этот факт, плюс аппаратное управление процессом сохранения и восстановления регистров делает на Эльбрусе невозможными атаки типа переполнения стека, когда за счёт переполнения на стек подставляется иной адрес возврата. Вообще, большое количество традиционных для x86/arm/mips хакерских атак на Э2K принципиально не работает. Отмечу - атаки не работают не в силу того, что хакеры пока не знают Э2K, а, как показано выше, в силу того что вообще невозможны в данной архитектуре.
- Конечно, нельзя гарантировать, что хакеры не найдут свои особенные пути взлома Э2K, но в целом он куда более защищен. Даже если не включать защищённый режим.

Защищенный режим

- Защищённый режим плох тем, что под него (пока?) не удалось собрать Линукс. В остальном - он великолепен.
- В защищённом режиме нельзя породить указатель из целого. Или из чего бы то ни было, кроме указателя.
- Кроме того - указатель имеет сложную структуру - база, размер и текущее смещение.
- Смещение не может выйти за размер. Адресная арифметика допустима в пределах [база, база+размер[- и этим же диапазоном ограничены указатели, которые мы можем породить из данного указателя.

Защищеный режим

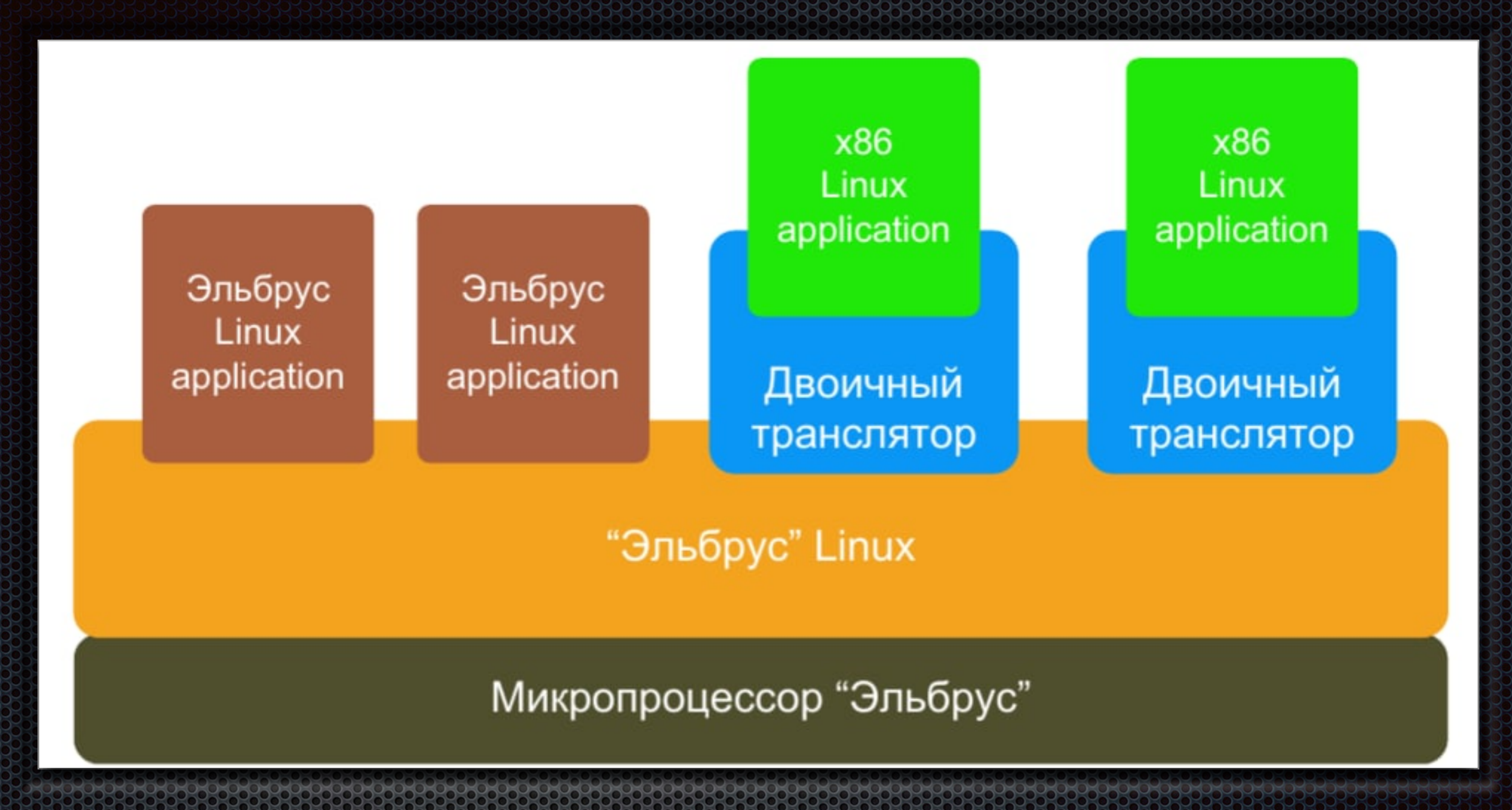
- Эффект от такого ограничения проще всего описать так: защищённость уровня Java для программ на Си. Managed memory на аппаратном уровне. Промазать указателем больше нельзя. Аппаратный valgrind на полной скорости.
- Нельзя просто так взять и просканировать память - вы можете обратиться только в ту часть памяти, к которой вам явно выдали указатель.
- Для реализации этого механизма процессор поддерживает теги - маркеры, которые говорят ему, что именно лежит в данной ячейке ОЗУ - указатель или данные.

Исполнение кода x86

- Процессор не умеет исполнять Интеловский код аппаратно, но для него реализована программная бинарная трансляция кода x86 в код Э2K. Трансляция выполняется прозрачно для исполняемого кода. Это позволяет запускать на Э2K операционные системы для Интеловских процессоров.
- Бинарная трансляция частично поддержана аппаратно - процессор содержит элементы аппаратуры архитектуры x86, которые очень накладно эмулировать программно - в частности, Э2K поддерживает сегментную адресацию и соответствующий набор сегментных регистров.

Бинарный транслятор

- Бинарный транслятор состоит из трёх версий генератора кода:
- Шаблонного (транслирует инструкции поштучно)
- Быстрого регионного (транслирует участки кода)
- Оптимизирующего - работает медленно, применяется для перекомпиляции очень горячих участков кода



Есть режим работы, который позволяет запускать скомпилированные под x86 Linux приложения на ядре Linux, скомпилированном нативно под Эльбрус.

Итог

- Э2K - инженерное чудо, один из самых интересных и современных процессоров сегодняшнего дня.
- Это не самый простой процессор, и для того, чтобы выжать из него полную скорость иногда нужно постараться.
- Я описал не всё. Только верхний уровень. Там ещё три этажа тонкостей и деталей. Не все детали прикладному программисту обязательно знать. Разработчики предоставляют довольно подробные учебные материалы для переноса кода на Э2K, а так же библиотеки, которые уже оптимизированы для этой архитектуры.
- Присоединяйтесь.

Вопросы

- Дмитрий Завалишин, dz@dz.ru
- DZ Systems group, http://dzsystems.com:
- Digital Zone, http://dz.ru
- E-legion, http://e-legion.com
- Aprentis, http://aprentis.ru

